New On-Line Preemptive Scheduling Policies for Improving Real-Time Behavior

Mathieu Grenier, Nicolas Navet

New On-Line Preemptive Scheduling Policies for Improving Real-Time Behavior

Mathieu Grenier, Nicolas Navet
LORIA-INRIA
Campus Scientifique, BP 239
54506 Vandoeuvre-lès-Nancy - France
{grenier, nnavet}@loria.fr
tel: +33 3 83 58 17 68, fax: +33 3 83 58 17 01

Abstract

In real-time systems, schedulability is mandatory but other application-dependent performance criteria are most generally of interest. We first define the properties that a "good" real-time scheduling algorithm must possess. Then, we exhibit a class of easily-implementable policies that should be well suited to various applicative contexts because, in our experiments, these policies provide good trade-off between feasibility and the satisfaction of the application-dependent criteria. We propose a schedulability analysis generic for all policies within this class and evaluate other criteria by simulation. The study is illustrated in the framework of computer-controlled systems that are known to be sensitive to various delays induced by resource sharing.

1 Introduction

Context of the paper. In real-time systems, feasibility of the task set is the basic requirement, but, usually, other criteria besides feasibility are of interest. A prominent example are computer-controlled systems [3] where it is well-known that other temporal characteristics than deadline respect affect the performances of the controlled system [24, 26, 25]. Classically, the design of a control loop assumes periodic executions and constant delays. In practice, once the control law is implemented, delays and variabilities arise, which leads to performance degradations and, sometimes, even jeopardizes the stability of the system.

Goal of our paper. The goal is here to find on-line scheduling policies that are well suited to the satisfaction of application dependent criteria whilst ensuring feasibility. In the following, we will illustrate our approach through computer-controlled systems where, most usually, reducing delays and their variabilities improve the performances.

Related Work. Many studies have been dedicated to find scheduling solutions that improve the performance of computer-controlled systems.

In [9], a modified version of the Constant Bandwidth Server (CBS), initially proposed in [1], is used to eliminate jitters. Data input and data output occur at fixed points in time and control tasks run in a CBS, which is an abstraction of a dedicated CPU offering a chosen fraction of the original CPU, to ensure that control tasks finish before the output of the data.

To better fit to the processing requirements of a control system, new task models have been conceived. In [6, 7], the elastic task model is proposed to handle overruns: task adapt their period at runtime in such a way as to keep the systems underloaded. In [11, 8], it is proposed that control tasks are subdivided in three different parts: sampling, computation, actuation. Sampling and actuation sub-tasks are assigned a high priority in order to reduce the jitters.

Another solution is to adjust the parameters of the tasks to achieve the desired goals. In [4], the worst-case end-of-execution jitter is minimized by choosing appropriate deadlines. In [12], initial offsets and priorities are adjusted to reduce jitter by minimizing preemption.

Improvements can also be brought by well choosing the parameters of the scheduling policies. In [13], a priority allocation scheme is proposed to reduce the average response time while, in [20], the problem of choosing scheduling policies and priorities on a Posix 1003.1b compliant operating system (OS) is tackled.

Finally, another way is to create new scheduling policies. In [2], the scheduler is synthetized as a timed automata from the Petri net modeling the system and the properties expected from the system. In [14], also starting from a Petri net model of the system, an optimal scheduling sequence is found by examining the marking graph of the Petri net.

Our approach. In this paper, we propose a technique for building new on-line scheduling policies that, on the one hand, ensure feasibility and, on the other hand, per-

form well with regard to application dependent criteria such as the ones that are crucial in computer-controlled systems. We do not merely tune the parameters of a scheduling policy, as the priorities [13] for FPP scheduling, but tune the scheduling algorithm itself. The main advantage with regard to [2] and [14], is that it scales well and is robust to modifications of the task sets which is almost unavoidable in an industrial design process. Furthermore, the implementation on off-the-shelf OS does not raise problem (see [15] for a prototype implementation on Posix1003.1b system). Finally, the approach could be used in conjunction with task splitting schemes [11, 8] or dedicated task models [6, 7]. Our proposal is made of three distinct steps:

1. define the characteristics that a "good" real-time scheduling policies must possess. This class of good policies constitutes the search space of our problem,
2. propose a schedulability analysis that is generic for all "good" policies,
3. explore the search space for finding policies that perform well in terms of feasibility and with respect to the other criteria.

Organization. In section 2, the model of the system and the assumptions made are presented. In section 3, we define the search space of the scheduling policies. Section 4 is dedicated to the "generic" feasibility analysis. In section 5, the criteria besides feasibility used in this study are introduced, and the experimental results are presented.

2 System model

This study deals with the non-idling scheduling of periodic tasks on a monoprocessor system. The tasks are not dependent (*i.e.* no precedence constraints) and their characteristics are known before run-time. In the following, the time is hypothesized to be discrete (*i.e.* durations are multiples of the clock time) which, in our context, is possible without loss of generality but implies that some care must be taken (see §4.2).

2.1 Task model

The task model is the classical one used in [18]. A periodic task τ_i is characterized by a triple $(C_i, \overline{D}_i, T_i)$ where C_i is the *Worst Case Execution Time* (WCET), $\overline{D}_i$ the *relative deadline* (*i.e.* maximum tolerable response time of an instance - equal for all instances of the same task) and T_i the *inter-arrival time* between two instances of τ_i. The release time of j^{th} instance of the i^{th} task is denoted by $A_{i,j}$. A *concrete* task $(\tau_i, A_{i,1})$ is a task for which the release time of its first instance $A_{i,1}$ is known before run-time while the release times of *non-concrete* tasks are unknown. For the sake of clarity, sporadic tasks and jitters in the availability dates are not considered here but can be taken into account as classically done.

Concrete and non-concrete tasks. In the following, a set of n periodic non-concrete tasks is denoted by $\Omega = \{\tau_1, \tau_2, ..., \tau_n\}$, where τ_i is a periodic task, while a set of n periodic concrete tasks is $\omega = \{(\tau_1, A_{1,1}), (\tau_2, A_{2,1}).....(\tau_n, A_{n,1})\}$, where $(\tau_i, A_{i,1})$ is a periodic concrete task. Without restrictions on the initial offsets, there is an infinite number of mapping from the set of non-concrete tasks Ω to the set of concrete tasks ω.

Feasibility and optimality. A concrete set of tasks ω is said *feasible* (or *schedulable*) if no instance of the system terminates its execution after its absolute deadline $D_{i,n} = A_{i,n} + \overline{D_i}$. A non-concrete set of tasks Ω is feasible, if all the concrete sets ω, which can be generated from Ω, are feasible.

A scheduling policy is *optimal* with respect to a certain criterion (e.g. feasibility, average response time) within its class if no other policy of the class performs better with respect to the criterion. In the following, *optimal* is used to say *optimal* with respect to feasibility. A scheduling policy is *non-concrete optimal* (with respect to feasibility) if it successfully schedules all the non-concrete sets that are schedulable with a policy of the class. As shown in [16], Earliest Deadline First (EDF) is non-concrete and concrete optimal within the class of non-idling policy. A policy is said *concrete optimal* if it schedules all the schedulable concrete sets.

2.2 Defining scheduling policies through priority functions

Priority functions is a convenient way of formally defining in a non-ambiguous manner scheduling policies, which, to our best knowledge, has been introduced for the first time in [19]. The priority function $\Gamma_{k,n}(t)$ indicates the priority of an instance $\tau_{k,n}$ at time t. The resource is assigned, at each time, according to the *Highest Priority First* (HPF) paradigm.

Function $\Gamma_{k,n}(t)$ takes its value from a totally ordered set $\mathcal{P}$ which is chosen in [19] to be the set of multi-dimensional $\mathbb{R}$-valued vectors $\mathcal{P} = \{(p_1, ..., p_n) \in \mathbb{R}^n \mid n \in \mathbb{N}\}$ provided with a lexicographical order. Between two vectors, coordinates are compared one by one starting from the left; the first different coordinate decides the priority order with the convention "the smaller the numerical value, the higher the priority". For instance, $\Gamma_{i,j}(t) = (3, 4, 5)$ and $\Gamma_{k,n}(t) = (3, 4, 6)$ implies that $\tau_{i,j}$ has a higher priority than $\tau_{k,n}$ at time t. Priority vectors of different sizes can be compared with the same rule as above and the convention that a missing coordinate is the lowest numerical value (e.g. $\Gamma_{i,j}(t) = (3, 4, 5)$ and $\Gamma_{k,n}(t) = (3, 4)$ means that $\tau_{k,n}$ has a higher priority than $\tau_{i,j}$ at time t). Finally, two vectors are equal iff they are the same size and if the components are equal one by one.

Most real-time scheduling policies can be defined easily using priority functions. For instance, the priority of an instance $\tau_{k,n}$ under preemptive EDF is $\Gamma^{EDF}_{k,n}(t) = (A_{k,n} + \overline{D}_k, k, n)$ (the last two coordinates are needed to

ensure decidability, see definition 2), Fixed Priority Preemptive with the Rate Monotonic (RM) priority assignment scheme is defined by $\Gamma_{k,n}^{FPP-RM}(t) = (T_k, k, n)$ and with Deadline Monotonic (DM) by $\Gamma_{k,n}^{FPP-DM}(t) = (\overline{D}_k, k, n)$. A class of policies of particular interest, to which EDF, FPP-RM and FPP-DM belong, are the *time independent policies*.

Definition 1 *[19] A scheduling policy $\mathcal{A}$ is time independent iff the priority of each instance does not vary over time:*

$$\forall t\, \Gamma_{k,n}^{\mathcal{A}}(t) = \Gamma_{k,n}^{\mathcal{A}}(0) = \Gamma_{k,n}^{\mathcal{A}}$$

Time independent policy are easily implementable since the priority of an instance is computed at release time and does not change anymore. Furthermore, context switches solely occur at arrival dates or when instances finish their execution.

Besides providing non-ambiguous definition of the scheduling policy, priority functions enable us to distinguish classes of scheduling policies and to derive generic results that are valid for whatever the policy belonging to a certain class. The next section presents the class of non-preemptive scheduling policy that will be studied in the rest of the paper.

3 Study domain

An arbitrary priority function does not necessarily define neither a scheduling of interest for real-time computing nor even a policy that can be implemented in practice. In this section, we precise the requirements expected from an acceptable policy (termed "good" policy in the following). Then, among the set of all good policies, we define the particular class of scheduling policies considered in this study.

3.1 "Good" scheduling policies

A "good" policy must meet a certain number of criteria, which are needed for the policy to be implemented in a real-time context.

Decidable policies. Policies are needed to be *decidable*: at any time t, there is exactly one instance of maximal priority among the set of active instances (*i.e.* instances with pending work). This concept of decidability was introduced in [19].

Definition 2 *[19] A priority function is decidable iff, at each time t such that work is pending, there is exactly one instance of maximal priority.*

For instance, the last two components of $\Gamma_{k,n}^{EDF}(t) = (A_{k,n} + \overline{D}_k, k, n)$ ensure decidability.

Implementable policies. For being *implementable* in practice, a policy must induce a finite number of context switches over a finite time interval. This first condition was exhibited in [19]. Furthermore, components of the priority vectors have to be representable by machine numbers.

Definition 3 *A scheduling policy is implementable iff the priority function:*

** is "piecewise order preserving": during any time interval of finite length, the number of changes of the highest priority instance is finite,*

** the coordinates of the priority function are "representable" by machines number.*

In the following, coordinates of a priority vector belong to the set of rational numbers $\mathbb{Q}$.

"Shift temporal invariant" policies. In this study, for the sake of predictability of the system, we are only interested in scheduling policies such that the relative priority between two instances does not depend on the numerical value of the clock: relative priority must remain the same if we "shift" the arrival of all instances to the left or the right. The policy is thus independent of the value of the system's clock at startup time. We call such policies *shift temporal invariant* (STI) policies. EDF is a STI policy since the priority between two instances only depends on the offset between arrival dates and on relative deadlines. On the contrary, a policy defined by $\Gamma_{k,n} = (C_k \cdot A_{k,n}, k, n)$ is not STI; just consider $\tau_{i,1}$ and $\tau_{j,1}$ with $A_{i,1} = 0$, $C_i = 10$ and $A_{j,1} = 1$ with $C_j = 1$ and the same two instances except that the arrival dates are shifted to the right by one unit of time.

Definition 4 *Let two concrete task sets be $\omega = \{(\tau_1, A_{1,1}), (\tau_2, A_{2,1}).....(\tau_n, A_{n,1})\}$ and $\omega' = \{(\tau_1, A_{1,1} + \Phi), (\tau_2, A_{2,1} + \Phi).....(\tau_n, A_{n,1} + \Phi)\}$ where ω' is a "shifted" version of ω (with $\Phi \in \mathbb{Z}$).*

A scheduling policy $\mathcal{A}$ is Shift Temporal Invariant (STI) iff for all possible Φ, $\forall i, j, k, n$ such that $(k, n) \neq (i, j)$ (two distinct instances), one has:

$$\begin{array}{rll} \forall t\, \Gamma_{k,n}^{\mathcal{A},\omega}(t) & \succ (\textit{resp} \prec) & \Gamma_{i,j}^{\mathcal{A},\omega}(t) \implies \\ \Gamma_{k,n}^{\mathcal{A},\omega'}(t+\Phi) & \succ (\textit{resp} \prec) & \Gamma_{i,j}^{\mathcal{A},\omega'}(t+\Phi) \end{array}$$

where $\Gamma_{k,n}^{\mathcal{A},\omega}(t)$ is the priority of $\tau_{k,n}$ of the concrete task set ω at time t.

We have defined a minimum set of requirements that a "good" policy must fulfill in the context of real-time computing: the policy must be decidable, implementable and shift temporal invariant. In the next paragraph, we precise the particular class of preemptive policies that will be studied in the rest of the paper.

3.2 Search space

In this study, we limit the search space to the class of "Arrival Time Dependent" policies. This choice is justified in the following.

"Arrival Time Dependent" policies. Our domain of study is a sub-class of Time Independent policies (see definition 1) that we call *Arrival Time Dependent Priority* (ATDP).

Definition 5 *An Arrival Time Dependent policy is a policy whose priority function can be put under the form*

$$\Gamma_{k,n}(t) = (A_{k,n} + p_k, k, n) \tag{1}$$

where p_k*:* $k \mapsto \mathbb{Q}$.

p_k is an arbitrary function, which returns a constant value for all instances of task τ_k. The value can be an arbitrary numerical value or it can be dependent of some characteristics of the task (*i.e.* $\overline{D}_k$, T_k and C_k). For instance, for EDF, p_k is equal to the relative deadline $\overline{D}_k$. Remark that policies, which have a priority function like $\Gamma_{k,n}(t) = (p_k, k, n)$, is the class of fixed-priority policies (FPP) and other optimization methods exists to attribute priority (see [13] for example).

Motivations for Arrival Time Dependent policies. First of all, ATD policies are "good" scheduling policies:

- decidability is ensured by the last two components of the priority vectors,
- the policies are implementable in the sense of definition 3; the priority functions are "piecewise order preserving" due to constant priority over time and they can be represented by machine numbers.

Secondly, ATD policies are promising in terms of performances. EDF belongs to this class but they may exist other policies that perform close to EDF in terms of feasibility while having a much better behavior with respect to application-dependent criteria, see §5.1 for example. The aim is to find scheduling policies that provide good trade-off between feasibility and other performance criteria of interest.

Thirdly, as it will be shown in §4.2, a generic feasibility analysis, through response time bounds, can be derived for all ATD policies.

4 Response time bounds for ATD policies

First, we recap the computation of bounds on response times for periodic tasks scheduled under EDF as initially proposed in [23]. Then, we show how this analysis can be extended for dealing with all Arrival Time Dependent policies.

4.1 EDF analysis: a recap

The response time $r_k(a)$ of a task instance is the time elapsed between its arrival a and its completion. The set of tasks is feasible under a given scheduling policy if the response time of each instance is lower or equal than the relative deadline. In general, it is not possible to compute the response times of all instances for all foreseeable trajectories of the system; a solution for assessing feasibility is to compute bounds on response times. Such an analysis was derived for preemptive EDF in [23].

In [23], it is shown that the worst case response time of an instance of a task occurs after a certain arrival pattern termed the "As Soon As Possible" pattern (ASAP for short). This result uses the concept of "deadline busy period" for an instance $\tau_{k,n}$, which is a period of processor utilization without idle-time during which only instances with deadline not greater than $\tau_{k,n}$ are executed.

Lemma 1 *[23] A response time bound of an instance of a task* τ_k *released* a *units of time after the beginning of its deadline busy period is found in a deadline busy period such as (ASAP pattern):*

- τ_k *has an instance released at time* a *(and possibly others released before),*
- *all others tasks are released from time* $t = 0$ *(beginning of the busy period) on at their maximum rate.*

This lemma allows to compute a bound on the response time $r_k(a)$ of an instance of τ_k released at time a, denoted $\tau_k(a)$ in the following. It was proven that the execution of $\tau_k(a)$ finishs, at the latest, at the time t solution of the following equation which can be solved by recurrence:

$$t = W_k(a,t) + \underbrace{\left(1 + \left\lfloor \frac{a}{\overline{T_k}} \right\rfloor\right) \cdot C_k}_{\substack{\text{work of instances} \\ \text{of tasks } \tau_k}} . \tag{2}$$

in which t is the length $L_k(a)$ of the "deadline busy period", and where $W_k(a,t)$ is an upper bound for the "higher priority workload" (i.e. work induced by instances of lower or equal deadlines) in an interval of length t :

$$W_k(a,t) = \sum_{i \neq k} \underbrace{\left(\min\left(\left\lceil \frac{t}{T_i} \right\rceil, \left\lfloor \frac{a + \overline{D_k} - \overline{D_i}}{T_i} \right\rfloor + 1\right)\right)^+}_{\substack{\text{maximum number of instances in interval } t \\ \text{with a deadline lower than or equal to } a + \overline{D}_k}} .C_i \tag{3}$$

and where the work of the instances of task τ_k is:

$$\left(1 + \left\lfloor \frac{a}{T_k} \right\rfloor\right) \cdot C_k$$

Thus, a bound on the response of an instance of $\tau_k(a)$ is:

$$r_k(a) = \max\left\{C_k, L_k(a) - a\right\}.$$

The response time bound for τ_k is $\max_a r_k(a)$. It is not possible to compute $r_k(a)$ for all values of a but it is proven in [23] that the only significants values of a are the elements of the set $\mathbf{A}_k$.

$$\mathbf{A}_k = \{t = n \times T_i + \overline{D}_i - \overline{D}_k \mid t \geq 0, t \leq \mathcal{L} - C_k, n \in \mathbb{N}, i = 1...m\}. \quad (4)$$

where $\mathcal{L}$ is the longest busy period (longest duration of the resource without idle time see [23] for computation details). In the next section, we show how this analysis can be easily adapted to Arrival Time Dependent policies.

4.2 Analysis for ATD policies

An instance $\tau_{k,n}$ under EDF possesses a priority vector equal to $(A_{k,n} + \overline{D}_k, k, n)$; EDF is thus a particular case of ATD policy where $p_k : k \mapsto \overline{D}_k$ (see definition of ATD policies in §3.2). In the following, it will be shown that Lemma 1 as well as the set of arrival dates to consider after the ASAP pattern (see equation 4) remain valid with the condition that $\overline{D}_k$ is replaced by p_k. "Deadline busy periods" become "priority busy periods" for an instance $\tau_{k,n}$ which are intervals of processor utilization without idle-time during which only instances with higher priority than $\tau_{k,n}$ are executed.

Lemma 2 *A response time bound of an instance of a task τ_k released a units of time after the beginning of its priority busy period is found in a priority busy period such as:*

- *τ_k has an instance released at time a (and possibly others released before),*
- *all other tasks are released from time $t = 0$ at their maximum rate.*

Sketch Of Proof:

Consider virtual-EDF, a modified version of EDF that would schedule tasks not by taking into account the actual relative deadline $\overline{D}_k$ but an arbitrary "virtual" deadline p_k . Its priority function is:

$$\Gamma_{k,n}^{virtual-EDF}(t) = (A_{k,n} + p_k,\ k,\ n),$$

where p_k, as $\overline{D}_k$, possesses the property that its value is equal for all instances of task τ_k. Indeed, this property on p_k is needed for lemma 4.1 in [23] to hold (precisely, when building the ASAP pattern, shifting left an instance must increase the higher priority workload).

According to lemma 1, a response time bound for τ_k under virtual-EDF occurs after the ASAP pattern, as defined by Spuri [23], where $\overline{D}_k$ is replaced by p_k in the equations 3 and 4. To assess the feasibility, the response time bounds just have to be compared with the actual relative deadlines $\overline{D}_k$.

■

The way to compute the worst case response time in nearly the same; only the value of $\overline{D}_k$ is replaced by p_k in equation 3.

Furthermore, the set $\mathbf{A}_k$ (*i.e.* the set of significant values of a where compute the response time), is (values of a which correspond to local maxima of $L_k(a) - a$):

$$\mathbf{A}_k = \{t = \lceil n \times T_i + p_i - p_k \rceil \mid t \geq 0, t \leq \mathcal{L} - C_k, n \in \mathbb{N}, i = 1...m\}. \quad (5)$$

Notice that p_i and p_k are rational values. Thus, $p_i - p_k \in \mathbb{Q}$ while, with the assumption of discrete time, the arrival times considered in $\mathbf{A}_k$ must belong to $\mathbb{N}$. In this case, the "significant" values of the EDF analysis (see equation 4) become here $\lceil n \times T_i + p_i - p_k \rceil$ (proof is given in Appendix A).

5 Experiments

Experiments in this study are performed in the framework of computer-controlled systems. Chosen performance criteria are presented in §5.1 while the space of scheduling policies that will be considered is defined in §5.2.

5.1 Performance criteria

We consider periodic control loops where the control algorithm is modeled by a periodic task τ_k with period T_k (i.e. the sampling period). In classical control theory, the main parts of a control loop are sampling, control computation and actuation. Some assumptions are made:

- the reading of data from sensors (i.e. sampling) is assumed to be done at the beginning of each instance (at time $B_{k,n}$),
- the computation of the control law is performed in a constant time C_k,
- the actuation, that is the transmission of output data to the actuators, is done at the end of execution of each task instance (at time $E_{k,n}$).

Specific delays of control loops have been identified to be of particular importance for the stability of the system, and, more generally, for its performances (see, for instance, studies in [3, 26, 25]). These delays are:

- *the input-output latency* of an instance $\tau_{k,n}$, which is the time elapsed between the sampling and the actuation. This value, denoted $iol_{k,n}$, is equal to $E_{k,n} - B_{k,n}$ with our notations,
- *the sampling interval* si_n, which is the time interval between two consecutive sampling instants (*i.e.* $B_{k,n+1} - B_{k,n}$),
- *the sampling latency* of an instance $\tau_{k,n}$, which is the time elapsed between the theoretical sampling time and its actual occurrence (*i.e.* $B_{k,n}$). This value $sl_{k,n}$ is equal to $B_{k,n} - A_{k,n}$,

In classical discrete control theory, input-output latencies and sampling intervals are assumed to be constant with no sampling latency. In practice, when resources are not dedicated to a single control loop, these delays exists and greatly impact the performances (see [25, 17]). The aim is thus to keep these delays and their variabilities (jitters) as close as possible to the assumptions made by the theory.

In the following, as to our best knowledge there is no analytic technique, values of the criteria are computed with the data collected during simulation runs. A given criterion is evaluated for a policy as the average value of the criterion for all tasks.

5.2 Search space

The aim is to find policies that performs good in terms of feasibility, for optimizing the use of resources, but also policies that are efficient with respect to the above-defined criteria. In the following, experiments will be done within a sub-class of ATDP policies having a priority vector of the form

$$\Gamma_{k,n} \quad = \quad (A_{k,n} + c.C_k + d.\overline{D_k}, k, n) \qquad (6)$$

where $d \in [0, 1]$ and $c \in [0, 100]$ (i.e. $p_k = c.C_k + d.\overline{D_k}$ in definition 5). A point $\mathcal{C}$ in our search space is a policy defined by a priority function having the form of equation 6.

This class has been chosen because we expect that it contains policies providing a good trade-off between feasibility and the satisfaction of the other criteria important for control systems (see 5.1). EDF actually belongs to this class and policies whose priority function is "close" to EDF are expected to have nearly the same behavior in terms of schedulability. On the other hand, introducing a term dependent of the execution time should help to improve the other criteria. Indeed, it has been shown that Shortest Remaining Processing Time First is optimal for average response times in various contexts (see [21, 22] quoted in [5]) and, in our experiments, Shortest Maximum Processing Time first (defined as $\Gamma_{k,n} = C_{k,n}$) performed much better than EDF for all defined criteria except, of course, feasibility.

5.3 Experimental results

We consider several control tasks sharing a CPU where the initial offsets of the tasks are not known (*i.e.* non-concrete set of tasks). In the following, we will distinguish the case where the policy has to be efficient on average (it can be used with different task sets) and the case where the policy is tuned for a particular application.

For the experiments, non-concrete task sets are generated with a global load randomly chosen in the interval $[0.8, 0.9]$ with $\overline{D_i} = T_i$. For the simulations, offsets are to be known; a concrete set of tasks is generated from a non-concrete one by randomly choosing the offset $A_{i,1}$ of each task τ_i in the interval $[0, T_i]$. Response time bounds and simulation software are implemented in C++; an applet version of the simulator is available at *http://www.loria.fr/equipes/TRIO/simulateur/SimApplet.html*.

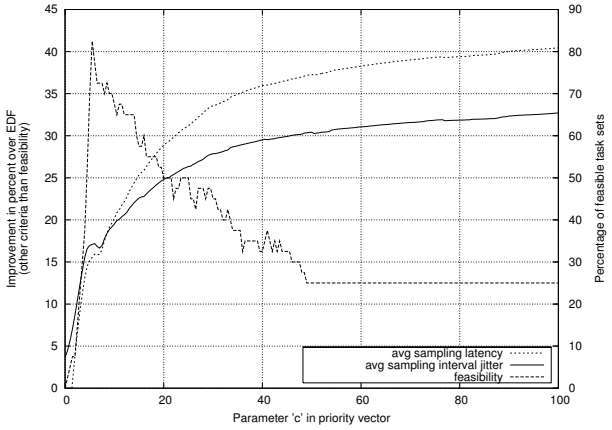

Figure 1. Feasibility, average sampling latency and average sampling interval jitter with comparison to EDF. The policies evaluated are defined by equation 6 with $d = 0.1$ and c ranging from 0 to 100. Experiments made on random sets of 10 tasks for an average load of 0.85.

5.3.1 Scheduling for the average case

Our search space is defined by equation 6. Figure 1 only shows the performances of the set of policies where $d = 0.1$ and where c takes its value in $[0, 100]$ with step 0.5 in equation 6. The two first performance criteria are the average sampling latency and the average sampling interval jitter (measured as the standard deviation of the sampling intervals), their values are read on the 'y' axis on the left and they are computed as the average value of 1500 simulation runs (100 non-concrete sets of tasks with 15 different offsets). The other criterion is feasibility; the 'y' axis on the right shows the percentage of feasible task sets where all task sets have been chosen to be feasible under plain EDF.

As expected, one sees on figure 1 that the larger the value of c in $\Gamma_{k,n}$, the better performances with respect to average sampling latency and average sampling interval jitter. The counterpart is that feasibility significantly diminishes when c increases. When c becomes large, terms $A_{k,n}$ and $d \cdot \overline{D}_k$ tend to be annihilated by $c \cdot C_k$ in equation 6 and policies behave in a quite similar manner as Shortest Maximum Processing Time first (see 5.2). The peak value for feasibility (i.e. c around 8) can be explained because, with the parameters of our experiments, $c \cdot C_k + d \cdot \overline{D}_{k,} \approx \overline{D}_k$ so the policies are close to EDF.

As an example, let us consider the policy defined by $\Gamma_{k,n} = (A_{k,n} + 15.C_k + \frac{1}{10}\overline{D}_k, k, n)$, it is feasible with 58% of task sets while achieving an improvement of 25% for the average sampling latency and of 22% for average

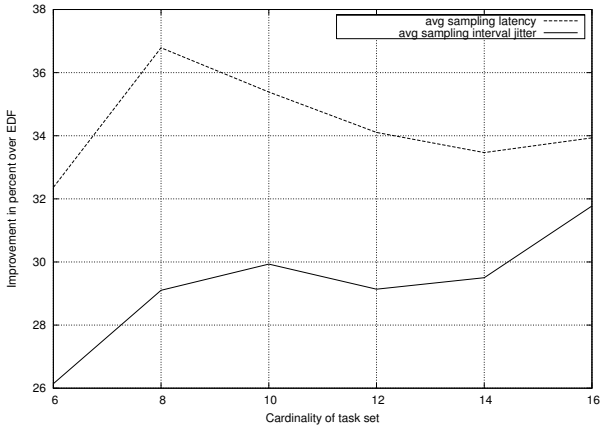

Figure 2. Average sampling latency and average sampling interval jitter with comparison to EDF for the best feasible policy found in the search space defined by equation 6. The improvement is evaluated for a number of task ranging from 6 to 16.

sampling interval jitter. In practice, a computer-controlled system will have better performances with this policy than with EDF. It worth noting that equivalent results, not shown here, were found for the average input-output latency and the input-output latency jitter.

5.3.2 Scheduling for a particular application

The aim is here to find the policy that leads to a feasible schedule and that provides the greatest improvement for the other criteria. The search space, defined by equation 6, is exhaustively searched with steps of granularity $d = 0.1$ and $c = 0.5$ (the search space comprises approximatively 2000 policies). Each point shown on figure 2 is the average improvement over 100 runs (only the best policy at each run is considered), where a run is defined by a task set randomly generated with an average load of 0.85. As in §5.3.1, the performance criteria are the average sampling latency and the average sampling interval jitter.

On figure 2, one sees that for a particular application, improvements are always larger than 32% for average sampling latency and larger than 26% for average sampling interval jitter whatever the cardinality of the set of tasks. For example, the average improvement achieved for 10 tasks is 35% for average sampling latency and 30% for average sampling interval jitter.

Overall, the technique is efficient, even on heavily loaded systems (average load of 0.85 in our experiments) and the improvement over EDF for average sampling latency and average sampling interval jitter is really significant whilst always ensuring feasibility. Similar results, not shown here, were found for the average input-output latency and the input-output latency jitter.

6 Conclusion and future work

In this paper, we highlight a class of on-line scheduling algorithms that are both easy to implement and that can provide interesting performances for feasibility and, especially, for other application-dependent criteria. We propose an algorithm to compute worst-case response time bounds that is generic for all policies of the class. Experiments show that, in the context of computer-controlled systems where delays and jitters impact the performances of the control loop, well chosen policies can bring important improvements over plain EDF.

In the future, we intend to evaluate more precisely the impact of the scheduling policies using software tools such as TrueTime [10] or the tool described in [17], that allow to integrate delays induced by the scheduling in the control loops. It is also planed to experiment new search techniques for exploring the policy search space; preliminary experiments show that simple neighbourhood techniques such hill-climbing are much more efficient than exhaustive search.

This work could be extended to other class of policies such as time-sharing policies (e.g. Round-Robin, Pfair). The main problem will be here to come up with a generic schedulability analysis.

References

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proc. of the 19th IEEE Real-Time Systems Symposium (RTSS 1998)*, 1998.

[2] K. Altisen, G. Goessler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *Proc. of the 20th IEEE Real-Time Systems Symphosium (RTSS 1999)*, 1999.

[3] K. Astrom and B. Wittenmark. *Computer-Controlled Systems*. Prentice Hall ISBN 0-13-168600-3, third edition, 1997.

[4] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari. Scheduling periodic task systems to minimize output jitter. In *Proc. of the 6th International Conference on Real-Time Computing Systems and Applications (RTCSA 1998)*, 1998.

[5] D. P. Bunde. SPT is optimally competitive for uniprocessor flow. *Information Processing Letters*, 90(5):233–238, 2004.

[6] G. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptative rate control. In *Proc. of the 19th IEEE Real-time Systems Symposium (RTSS 1998)*, 1998.

[7] M. Caccamo, G. Buttazzo, and L. Sha. Elastic feedback control. In *Proc. of the 12th Euromicro Conference on Real-Time Systems (Euromicro-RTS 2000)*, 2000.

[8] A. Cervin. *Integrated Control and Real-Time Scheduling*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, 2003.

[9] A. Cervin and J. Eker. Control-scheduling codesign of real-time systems: The control server approach. *Journal of Embedded Computing*, 1(2), 2004.

[10] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén. How does control timing affect performance? *IEEE Control Systems Magazine*, 23(3):16–30, 2003.

[11] A. Crespo, I. Ripoll, and P. Albertos. Reducing delays in RT control: the control action interval. In *Proc. of the 14th IFAC World Congress*, 1999.

[12] L. David, F. Cottet, and N. Nissanke. Jitter control in on-line scheduling of dependent real-time tasks. In *Proc. of the 22nd IEEE Real-time Systems Symposium (RTSS 2001)*, 2001.

[13] J. Goossens and P. Richard. Performance optimization for hard real-time fixed priority tasks. In *Proc. of the 12th international conference on real-time systems (RTS 2004)*, 2004.

[14] E. Grolleau, A. Choquet-Cheniet, and F. Cottet. Validation de systèmes temps réel à l'aide de réseaux de petri. In *Proc. of Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'98)*, 1998.

[15] S. Gury and M. Glasson. Implémentation d'un ordonnanceur de tâches à fonction de priorités sur Posix 1003.1b. 2nd year project of Ecole des Mines, 2004. Available at http://www-eleves-isia.cma.fr/

[16] K. Jeffay, D. Stanat, and C. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proc. of the 12th IEEE Real-time Systems Symposium (RTSS 1991)*, 1991.

[17] F. Jumel, N. Navet, and F. Simonot-Lion. Evaluation de la qualité de fonctionnement d'une application de contrôle-commande en fonction de caractéristiques de son implantation. *to appear in Technique et Science Informatiques*, 2005.

[18] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard-real time environnement. *Journal of the ACM*, 20(1):40–61, 1973.

[19] J. Migge. *Scheduling under Real-Time Constraints: a Trajectory Based Model*. PhD thesis, University of Nice Sophia-Antipolis, 1999.

[20] N. Navet and J. Migge. Fine tuning the scheduling of tasks through a genetic algorithm: Application to Posix1003.1b compliant OS. In *Proc. of IEE Proceedings Software*, volume 150, pages 13–24, 2003.

[21] L. Schrage. A proof of the optimality of the shortest remaining time discipline. *Operations Research*, 16:687–690, 1968.

[22] D. Smith. A new proof of the optimality of the shortest remaining time discipline. *Operations Research*, 26(1):197–199, 1976.

[23] M. Spuri. Analysis of deadline scheduling in real-time systems. Technical Report RR-2772, INRIA, 1996. Available at http://www.inria.fr/rrrt/rr-2772.html.

[24] M. Törngren. Fundamentals of implementing real-time control applications in distributed computer systems. *Real-Time Systems*, 14(3):219–250, 1998.

[25] B. Wittenmark, J. Nilsson, and M. Törngren. Timing problems in real-time control systems. In *Proc. of the American Control Conference*, 1995.

[26] K.-E. Årzén, A. Cervin, J. Eker, and L. Sha. An introduction to control and scheduling co-design. In *Proc. of the 39th IEEE Conference on Decision and Control*, 2000.

A Significant values of a for response time bound analysis

A proof of formula 5 is given here.

Which a are to be analyzed ? Only arrival dates a, which imply changes in the workload brought by the others tasks and by the same task, are to be considered. Indeed, let a_1 and a_2 be two arrival dates with $a_2 > a_1$ and

$$W_k(a_1,t)+\left(1+\left\lfloor\frac{a_1}{T_k}\right\rfloor\right)\cdot C_k = W_k(a_2,t)+\left(1+\left\lfloor\frac{a_2}{T_k}\right\rfloor\right)\cdot C_k\,, \tag{7}$$

then $r_k(a_1) \geq r_k(a_2)$. Indeed, from equation 7 and equation 2:

$$L_k(a_1) = L_k(a_2)$$
$$L_k(a_1) - a_1 \geq L_k(a_2) - a_2$$
$$r_k(a_1) \geq r_k(a_2).$$

Thus, only the response time for the instance arrived in a_1 has to be computed.

Significant values of a. Let us determine the values of a that induce changes in the workload

1. brought by instances of the same task (*i.e.* $\left(1+\left\lfloor\frac{a}{T_k}\right\rfloor\right)\cdot C_k$):

   $\forall n \in \mathbb{N}$ then $\left\lfloor\frac{a}{T_k}\right\rfloor = n$ iff $n\cdot T_k \leq a < (n+1)\cdot T_k$. The values of a that imply changes are:

   $$a \in \{t = n\cdot T_k \mid t \geq 0, n \in \mathbb{N}\}. \tag{8}$$

2. brought by instances of the others tasks (*i.e.* $W_k(a,t) = \sum_{i\neq k} \underbrace{\left(\min\left(\left\lceil\frac{t}{T_i}\right\rceil, \left\lfloor\frac{a+p_k-p_i}{T_i}\right\rfloor+1\right)\right)^+}_{\text{maximum number of instances in interval } t}\cdot C_i$):

   $\forall n \in \mathbb{N}, \forall i \neq k$ then $\left\lfloor\frac{a+p_k-p_i}{T_i}\right\rfloor = n$ iff:

   $$n \leq \frac{a+p_k-p_i}{T_i} < n+1,$$

   $$n\cdot T_i + p_i - p_k \leq a < (n+1)\cdot T_i + p_i - p_k.$$

   as by assumption $a \in \mathbb{N}$, and $n\cdot T_i + p_i - p_k \in \mathbb{Q}$, then the smallest value of a greater than $n\cdot T_i + p_i - p_k$ which is by definition $\lceil n\times T_i + p_i - p_k\rceil$. The values of a that imply changes in $\left\lfloor\frac{a+p_k-p_i}{T_i}\right\rfloor$ are thus

   $$a \in \{t = \lceil n\times T_i + p_i - p_k\rceil \mid t>0, i=\{1,2,...,m\}\backslash\{k\}, n\in\mathbb{N}\}. \tag{9}$$

Finally, from equation 8 and equation 9, the values of a for which response time bounds have to be computed are:

$$a \in \{t = \lceil n\times T_i + p_i - p_k\rceil \mid t>0, t<\mathcal{L}-C_k, i=1..m, n\in\mathbb{N}\}.$$